Tobacco use and mortality, 2004-2015

1. Introduction

Smoking remains one of the leading causes of preventable diseases and deaths worldwide. Despite public health efforts, millions still suffer the consequences of tobacco use, including cancer, respiratory, and circulatory diseases. This project aims to build a data-driven understanding of how smoking prevalence, government intervention (like prescription support), and economic factors (such as affordability) impact smoking-related fatalities**.**

2. Purpose and Objective

2.1 Purpose:

To analyze multi-source datasets on tobacco usage, cessation support, healthcare burden, and economic factors to:

- Understand trends in smoking and health impacts.
- Predict tobacco-related fatalities using machine learning.

2.2 Objectives:

- Aggregate and clean health and economic datasets.
- Perform feature engineering for meaningful modeling.
- Visualize key relationships and temporal trends.
- Build a predictive model to estimate fatalities.
- Interpret feature influence and recommend policies.

3. Datasets Used

The project combines five datasets, each contributing unique insights:

Dataset	Description
smokers_df	Smoking prevalence by age and year
prescriptions_df	Use of nicotine replacement and cessation drugs
metrics_df	Economic indicators like tobacco affordability
fatalities_df	Number of deaths from smoking-related diseases
admissions_df	Hospital admissions due to smoking-linked conditions

4. Code Explanation

4.1 Import Libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from xgboost import XGBRegressor
import matplotlib.pyplot as plt
```

- Pandas is a powerful library for data manipulation and analysis**.**
- NumPy is the core library for numerical operations**.**
- Imports train_test_split() from Scikit-learn.
- Imports evaluation metrics:
    - mean_squared_error: Measures prediction error.
    - r2_score: Indicates how well the model fits the data (1 is perfect, 0 means no fit).
- Imports XGBRegressor from the XGBoost library.
- XGBoost is an efficient gradient boosting framework used for regression and classification**.**
- Imports the Matplotlib plotting library.

4.2 Load data

```
# Load datasets
smokers_df = pd.read_csv("/content/smokers.csv")
prescriptions_df = pd.read_csv("/content/prescriptions.csv")
metrics_df = pd.read_csv("/content/metrics.csv")
fatalities_df = pd.read_csv("/content/fatalities.csv")
admissions_df = pd.read_csv("/content/admissions.csv")
```

4.3 Display first few rows of each dataset to understand structure

```
smokers_df.head()
```

	Year	Method	Sex	16 and Over	16-24	25-34	35-49	50-59	60 and Over
0	1974	Unweighted	NaN	46	44	51	52	50	33
1	1976	Unweighted	NaN	42	42	45	48	48	30
2	1978	Unweighted	NaN	40	39	45	45	45	30
3	1980	Unweighted	NaN	39	37	46	44	45	29
4	1982	Unweighted	NaN	35	35	38	39	41	27

```
prescriptions_df.head()
```

	Year	All Pharmacotherapy Prescriptions	Nicotine Replacement Therapy (NRT) Prescriptions	Bupropion (Zyban) Prescriptions	Varenicline (Champix) Prescriptions	Net Ingredient Cost of All Pharmacotherapies	Net Ingredient Cost of Nicotine Replacement Therapies (NRT)	Net Ingredient Cost of Bupropion (Zyban)	Net Ingredient Cost of Varenicline (Champix)
0	2014/15	1348	766	21	561.0	38145	18208	807	19129.0
1	2013/14	1778	1059	22	697.0	48767	24257	865	23646.0
2	2012/13	2203	1318	26	859.0	58121	28069	994	29058.0
3	2011/12	2532	1545	30	957.0	64552	30951	1216	32385.0
4	2010/11	2564	1541	36	987.0	65883	30808	1581	33494.0

```
metrics_df.head()
```

	Year	Tobacco Price\nIndex	Retail Prices\nIndex	Tobacco Price Index Relative to Retail Price Index	Real Households' Disposable Income	Affordability of Tobacco Index	Household Expenditure on Tobacco	Household Expenditure Total	Expenditure on Tobacco as a Percentage of Expenditure
0	2015	1294.3	386.7	334.7	196.4	58.7	19252.0	1152387.0	1.7
1	2014	1226.0	383.0	320.1	190.0	59.4	19411.0	1118992.0	1.7
2	2013	1139.3	374.2	304.5	190.3	62.5	18683.0	1073106.0	1.7
3	2012	1057.8	363.1	291.3	192.9	66.2	18702.0	1029378.0	1.8
4	2011	974.9	351.9	277.1	189.3	68.3	18217.0	990828.0	1.8

```
fatalities_df.head()
```

	Year	ICD10 Code	ICD10 Diagnosis	Diagnosis Type	Metric	Sex	Value
0	2014	All codes	All deaths	All deaths	Number of observed deaths	NaN	459087
1	2014	C33-C34 & C00-C14 & C15 & C32 & C53 & C67 & C6...	All deaths which can be caused by smoking	All deaths which can be caused by smoking	Number of observed deaths	NaN	235820
2	2014	C00-D48	All cancers	All cancers	Number of observed deaths	NaN	136312
3	2014	J00-J99	All respiratory diseases	All respiratory diseases	Number of observed deaths	NaN	61744
4	2014	I00-I99	All circulatory diseases	All circulatory diseases	Number of observed deaths	NaN	126101

```
admissions_df.head()
```

	Year	ICD10 Code	ICD10 Diagnosis	Diagnosis Type	Metric	Sex	Value
0	2014/15	All codes	All admissions	All admissions	Number of admissions	NaN	11011882
1	2014/15	C33-C34 & C00-C14 & C15 & C32 & C53 & C67 & C6...	All diseases which can be caused by smoking	All diseases which can be caused by smoking	Number of admissions	NaN	1713330
2	2014/15	C00-D48	All cancers	All cancers	Number of admissions	NaN	1691035
3	2014/15	J00-J99	All respiratory diseases	All respiratory diseases	Number of admissions	NaN	611002
4	2014/15	I00-I99	All circulatory diseases	All circulatory diseases	Number of admissions	NaN	907157

4.4 Check Missing Values in the Dataset

```python
# Check for missing values in each dataset
print(smokers_df.isnull().sum())
print(prescriptions_df.isnull().sum())
print(metrics_df.isnull().sum())
print(fatalities_df.isnull().sum())
print(admissions_df.isnull().sum())
```

```
Year            0
Method          0
Sex            28
16 and Over     0
16-24           0
25-34           0
35-49           0
50-59           0
60 and Over     0
dtype: int64
Year                                                    0
All Pharmacotherapy Prescriptions                       0
Nicotine Replacement Therapy (NRT) Prescriptions        0
Bupropion (Zyban) Prescriptions                         0
Varenicline (Champix) Prescriptions                     2
Net Ingredient Cost of All Pharmacotherapies            0
Net Ingredient Cost of Nicotine Replacement Therapies (NRT)    0
Net Ingredient Cost of Bupropion (Zyban)                0
Net Ingredient Cost of Varenicline (Champix)            2
dtype: int64
Year                                                  0
Tobacco Price\nIndex                                  0
Retail Prices\nIndex                                  0
Tobacco Price Index Relative to Retail Price Index    0
Real Households' Disposable Income                    0
Affordability of Tobacco Index                        0
Household Expenditure on Tobacco                      5
Household Expenditure Total                           5
Expenditure on Tobacco as a Percentage of Expenditure    5
dtype: int64
Year                0
ICD10 Code          0
ICD10 Diagnosis     0
Diagnosis Type      0
Metric              0
Sex               583
Value               0
dtype: int64
Year                0
ICD10 Code          0
ICD10 Diagnosis     0
Diagnosis Type      0
Metric              0
Sex               693
Value               1
dtype: int64
```

4.5 Check the column names for any inconsistencies

```
# Check the column names for any inconsistencies
print(smokers_df.columns)
print(prescriptions_df.columns)
print(metrics_df.columns)
print(fatalities_df.columns)
print(admissions_df.columns)
```

```
Index(['Year', 'Method', 'Sex', '16 and Over', '16-24', '25-34', '35-49',
       '50-59', '60 and Over'],
      dtype='object')
Index(['Year', 'All Pharmacotherapy Prescriptions',
       'Nicotine Replacement Therapy (NRT) Prescriptions',
       'Bupropion (Zyban) Prescriptions',
       'Varenicline (Champix) Prescriptions',
       'Net Ingredient Cost of All Pharmacotherapies',
       'Net Ingredient Cost of Nicotine Replacement Therapies (NRT)',
       'Net Ingredient Cost of Bupropion (Zyban)',
       'Net Ingredient Cost of Varenicline (Champix)'],
      dtype='object')
Index(['Year', 'Tobacco Price\nIndex', 'Retail Prices\nIndex',
       'Tobacco Price Index Relative to Retail Price Index',
       'Real Households' Disposable Income', 'Affordability of Tobacco Index',
       'Household Expenditure on Tobacco', 'Household Expenditure Total',
       'Expenditure on Tobacco as a Percentage of Expenditure'],
      dtype='object')
Index(['Year', 'ICD10 Code', 'ICD10 Diagnosis', 'Diagnosis Type', 'Metric',
       'Sex', 'Value'],
      dtype='object')
Index(['Year', 'ICD10 Code', 'ICD10 Diagnosis', 'Diagnosis Type', 'Metric',
       'Sex', 'Value'],
      dtype='object')
```

4.6 Preprocess: Fix Year formats

```
for df in [prescriptions_df, admissions_df]:
    df['Year'] = df['Year'].astype(str).str.extract(r'(\d{4})').astype(int)

# Clean whitespace in column names
metrics_df.columns = metrics_df.columns.str.strip()
fatalities_df.columns = fatalities_df.columns.str.strip()
admissions_df.columns = admissions_df.columns.str.strip()
```

This Code Explains:

```
for df in [prescriptions_df, admissions_df]:
    df['Year'] = df['Year'].astype(str).str.extract(r'(\d{4})').astype(int)
```

- It loops through two DataFrames: prescriptions_df and admissions_df.
- For each DataFrame, it cleans the Year column using these steps:
    - **astype(str)**
        - Converts the Year column to a string (in case it's not already).

- **.str.extract(r'(\d{4})')**
  - Uses regular expression to extract a 4-digit number (i.e., a proper year like 2014). This is useful if the year was in a format like '2014/15' or '2014-15'.
- **.astype(int)**
  - Converts the extracted 4-digit year to an integer type.

```
# Clean whitespace in column names
metrics_df.columns = metrics_df.columns.str.strip()
fatalities_df.columns = fatalities_df.columns.str.strip()
admissions_df.columns = admissions_df.columns.str.strip()
```

- Removes leading and trailing whitespace from column names in these DataFrames.

4.7 Aggregate each dataset

```
#Smokers: avg across age groups
age_cols = ['16 and Over', '16-24', '25-34', '35-49', '50-59', '60 and Over']
smokers_df[age_cols] = smokers_df[age_cols].apply(pd.to_numeric, errors='coerce')
smokers_df['Avg_Smoking_Rate'] = smokers_df[age_cols].mean(axis=1)
smoker_summary = smokers_df.groupby('Year')['Avg_Smoking_Rate'].mean().reset_index()
```

- Converts age-specific columns to numeric values.
- Computes the average smoking rate across all age groups for each row.
- Groups by Year and takes the average smoking rate for each year.

Result: smoker_summary → Yearly average smoking rate across all age groups.

```
# --- Prescriptions: total prescriptions
prescription_cols = [
    'All Pharmacotherapy Prescriptions',
    'Nicotine Replacement Therapy (NRT) Prescriptions',
    'Bupropion (Zyban) Prescriptions',
    'Varenicline (Champix) Prescriptions'
]
prescriptions_df[prescription_cols] = prescriptions_df[prescription_cols].apply(pd.to_numeric, errors='coerce')
prescriptions_df['Total_Prescriptions'] = prescriptions_df[prescription_cols].sum(axis=1)
prescription_summary = prescriptions_df[['Year', 'Total_Prescriptions']]
```

- Converts prescription-related columns to numbers.
- Sums all prescription types into Total_Prescriptions for each row.
- Keeps only Year and Total_Prescriptions columns.

Result: prescription_summary → Yearly total count of smoking cessation prescriptions.

```
# --- Metrics: keep selected columns
metrics_summary = metrics_df[[
    'Year',
    'Tobacco Price Index Relative to Retail Price Index',
    'Affordability of Tobacco Index',
    'Expenditure on Tobacco as a Percentage of Expenditure'
]].copy()
metrics_summary.columns = ['Year', 'Price_Index_Relative', 'Affordability_Index', 'Tobacco_Spending_Percent']
```

- Selects only relevant columns: price index, affordability, and tobacco spending.
- Renames the columns to cleaner, analysis-friendly names.

Result: metrics_summary → Yearly economic factors affecting tobacco consumption.

```
# --- Fatalities: filter for "All deaths which can be caused by smoking"
fatal_df = fatalities_df[
    fatalities_df['Diagnosis Type'] == 'All deaths which can be caused by smoking'
]
fatal_df['Value'] = pd.to_numeric(fatal_df['Value'], errors='coerce')
fatal_summary = fatal_df.groupby('Year')['Value'].sum().reset_index()
fatal_summary.columns = ['Year', 'Fatalities']
```

- Filters only rows where deaths are caused by smoking.
- Converts the death count (Value) to numeric.
- Groups by year and sums the fatalities.

Result: fatal_summary → Yearly count of smoking-related deaths.

```
# --- Admissions: filter for "All diseases which can be caused by smoking"
adm_df = admissions_df[
    admissions_df['Diagnosis Type'] == 'All diseases which can be caused by smoking'
]
adm_df['Value'] = pd.to_numeric(adm_df['Value'], errors='coerce')
admission_summary = adm_df.groupby('Year')['Value'].sum().reset_index()
admission_summary.columns = ['Year', 'Total_Admissions']
```

- Filters for diseases caused by smoking.
- Converts values to numeric.
- Groups by year and sums total admissions.

Result: admission_summary → Yearly count of smoking-related hospital admissions.

4.8 Merge datasets

```
df = fatal_summary \
     .merge(smoker_summary, on='Year', how='left') \
     .merge(prescription_summary, on='Year', how='left') \
     .merge(metrics_summary, on='Year', how='left') \
     .merge(admission_summary, on='Year', how='left')

df.dropna(inplace=True)
```

- **fatal_summary:**
  - This is the main dataset containing the target variable: Fatalities (smoking-related deaths per year).
- Merging the rest of the summaries on the Year column, one by one:
  - smoker_summary: Adds Avg_Smoking_Rate per year.
  - prescription_summary: Adds Total_Prescriptions per year.
  - metrics_summary: Add three columns - Price_Index_Relative, Affordability_Index, and Tobacco_Spending_Percent.
  - admission_summary: Adds Total_Admissions per year.
  - All merges are **how='left'**, which means:
    - Keep all rows from fatal_summary (even if data is missing in others).
    - Merge the others where their Year matches.
- **df.dropna(inplace=True):**
  - After merging, any year that had missing data in any of the merged datasets will have NaN values.
  - This line drops all such rows, ensuring the final dataset (df) is complete and clean, with no missing values — which is critical for training machine learning models.

4.9 Model: XGBoost Regressor

```
features = ['Avg_Smoking_Rate', 'Total_Prescriptions', 'Price_Index_Relative',
            'Affordability_Index', 'Tobacco_Spending_Percent', 'Total_Admissions']
target = 'Fatalities'

X = df[features]
y = df[target]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
model.fit(X_train, y_train)
```

This Code Explains:

```
features = ['Avg_Smoking_Rate', 'Total_Prescriptions', 'Price_Index_Relative',
            'Affordability_Index', 'Tobacco_Spending_Percent', 'Total_Admissions']
target = 'Fatalities'
```

- **features:** These are the independent variables (inputs) used to predict.

- **target:** This is the dependent variable (output) — Fatalities.

```
X = df[features]
y = df[target]
```

- **X:** DataFrame containing only the feature columns.
- **y:** Series containing the target variable.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- This divides the dataset:
    - 80% for training the model.
    - 20% for testing the model's performance.
- **random_state=42:** Ensures reproducibility — same split each time.

```
model = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
model.fit(X_train, y_train)
```

- **XGBRegressor:** A powerful, gradient-boosting regression algorithm known for high performance.
- **Parameters:**
    - **n_estimators=100:** Model builds 100 decision trees (iterations).
    - **learning_rate=0.1:** Step size for each boosting step (controls model complexity).
    - **random_state=42:** Again, for reproducibility.
- **fit():** Trains the model using the training data.

Result: trained model (model) that has learned patterns in the data and can predict smoking-related fatalities based on the features provided.

4.10 Evaluation

```
[18] y_pred = model.predict(X_test)
     print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
     print("R² Score:", r2_score(y_test, y_pred))
```

```
Mean Squared Error: 264796480.0
R² Score: 0.8347970247268677
```

This Code Explains:

```
y_pred = model.predict(X_test)
```

- This line uses the trained model to predict fatalities (y_pred) based on the unseen test features (X_test).

```
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
```

- Mean Squared Error (MSE) measures the average squared difference between actual values (y_test) and predicted values (y_pred).
- Lower MSE is better, and it indicates that predictions are close to the actual data.
- R² Score (Coefficient of Determination) tells how much of the variation in the target variable can be explained by the features.
- Range:
    - = Perfect prediction.
    - 0 = Model does no better than average.
    - Negative = Model is worse than just predicting the mean.

```
Mean Squared Error: 264796480.0
R² Score: 0.8347970247268677
```

Result:

- On average, the model's predictions deviate from the true fatality values by a large margin, because the values being predicted are likely in the millions range.
- The model explains 83.5% of the variance in smoking-related fatalities.
- This is considered a strong predictive performance, suggesting that the features used (like smoking rate, affordability index, etc.) are good predictors of fatalities.

4.11 Plot: Actual vs Predicted

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='navy', alpha=0.7)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.xlabel("Actual Fatalities")
plt.ylabel("Predicted Fatalities")
plt.title("Tobacco-related Fatalities: Actual vs Predicted")
plt.grid(True)
plt.tight_layout()
plt.show()
```

This Code Explains:

```
plt.figure(figsize=(8, 6))
```

- Creates a new plot with a size of 8x6 inches.

```
plt.scatter(y_test, y_pred, color='navy', alpha=0.7)
```

- Plots a scatter plot of actual (y_test) vs predicted (y_pred) fatalities.
- Points are navy blue and slightly transparent (alpha=0.7).

```
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
```

- Draws a red dashed line representing the ideal case where actual = predicted.

```
plt.xlabel("Actual Fatalities")
plt.ylabel("Predicted Fatalities")
```

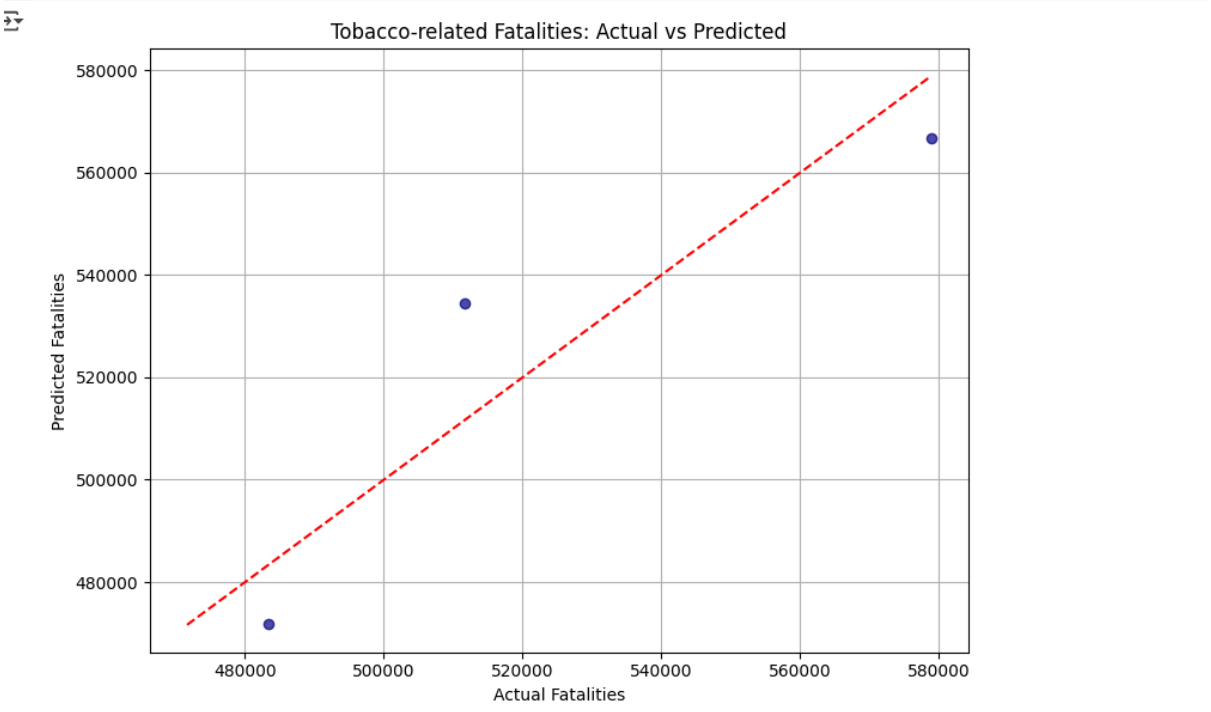
- Labels the X and Y axes.

```
plt.title("Tobacco-related Fatalities: Actual vs Predicted")
```

- Sets the plot title.

```
plt.grid(True)
plt.tight_layout()
plt.show()
```

- Adds grid lines, adjusts layout to avoid clipping, and displays the plot.



Result:

- Blue dots represent actual vs predicted values.
- The closer these dots are to the red dashed line, the better the model's predictions.

- If a dot is above the red line, it means the prediction is higher than actual.
- If a dot is below the red line, the prediction is lower than actual.
- Predictions are relatively close, but there's some deviation (especially the lower-left point), suggesting some prediction error.
- The model seems to be moderately accurate, but not perfect

4.12 Plot: Feature Importance

```
importances = model.feature_importances_
sorted_idx = np.argsort(importances)

plt.figure(figsize=(8, 5))
plt.barh(range(len(importances)), importances[sorted_idx], color='teal')
plt.yticks(range(len(importances)), [features[i] for i in sorted_idx])
plt.title("XGBoost Feature Importance")
plt.xlabel("Importance Score")
plt.tight_layout()
plt.show()
```

This Code Explains:

```
importances = model.feature importances
```

- Retrieves the importance scores of each feature from the trained XGBoost model.

```
sorted_idx = np.argsort(importances)
```

- Sorts the feature indices based on importance scores in ascending order (for better visualization).

```
plt.figure(figsize=(8, 5))
plt.barh(range(len(importances)), importances[sorted_idx], color='teal')
```

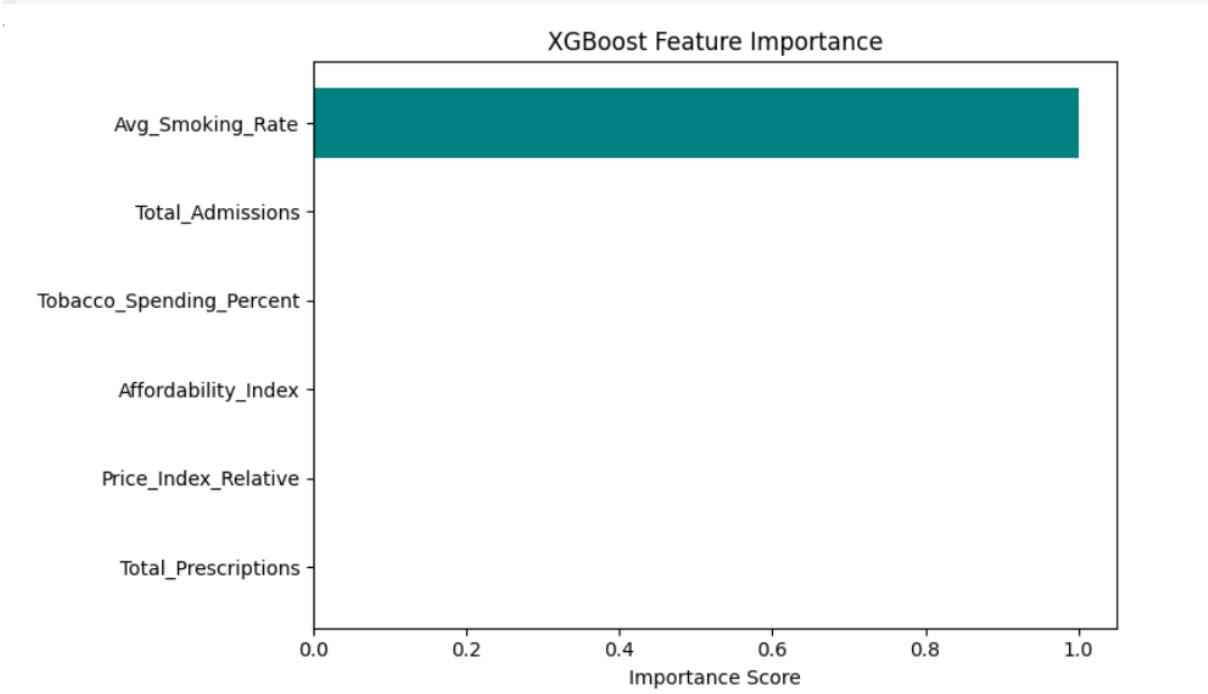
- Plots a horizontal bar chart of feature importances.
- Bars are colored teal.

```
plt.yticks(range(len(importances)), [features[i] for i in sorted_idx])
```

- Labels the Y-axis ticks with the actual feature names, sorted by importance.

```
plt.title("XGBoost Feature Importance")
plt.xlabel("Importance Score")
plt.tight_layout()
plt.show()
```

- Adds a title and axis label, ensures a neat layout, and displays the plot.



Result:

- Avg_Smoking_Rate has almost all the importance, suggesting that the model's predictions are almost entirely based on this single feature.
- All other features (Total_Admissions, Tobacco_Spending_Percent, Affordability_Index, etc.) have near-zero importance, meaning the model found them not useful for predicting the target.

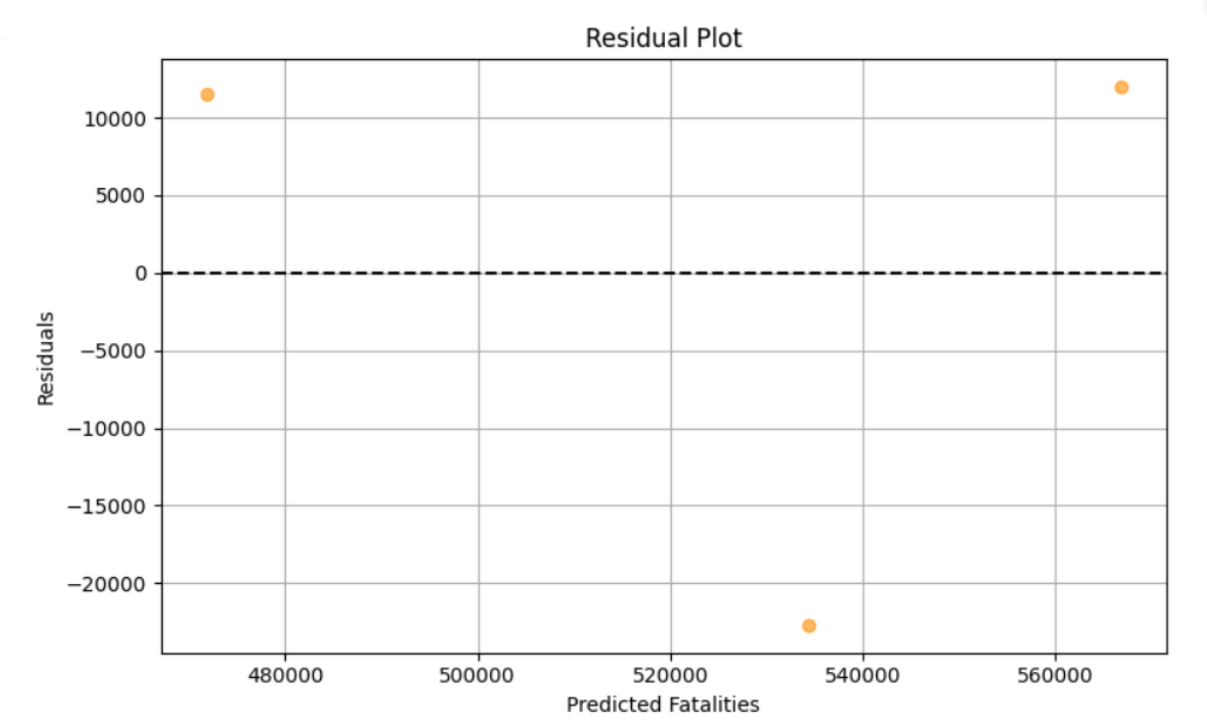
4.13 Plot: Residuals

```
residuals = y_test - y_pred
plt.figure(figsize=(8, 5))
plt.scatter(y_pred, residuals, alpha=0.6, color='darkorange')
plt.axhline(0, color='black', linestyle='--')
plt.title("Residual Plot")
plt.xlabel("Predicted Fatalities")
plt.ylabel("Residuals")
plt.grid(True)
plt.tight_layout()
plt.show()
```

This Code Explains:

- This is a residual plot, where:
  - X-axis = predicted values from your model (y_pred)
  - Y-axis = residuals (y_test - y_pred), i.e., how far off the prediction is

Result:



The residuals vary widely:

- One point has a very large negative residual (model overpredicted by ~23,000).
- Two points have large positive residuals (model underpredicted by ~11,000).

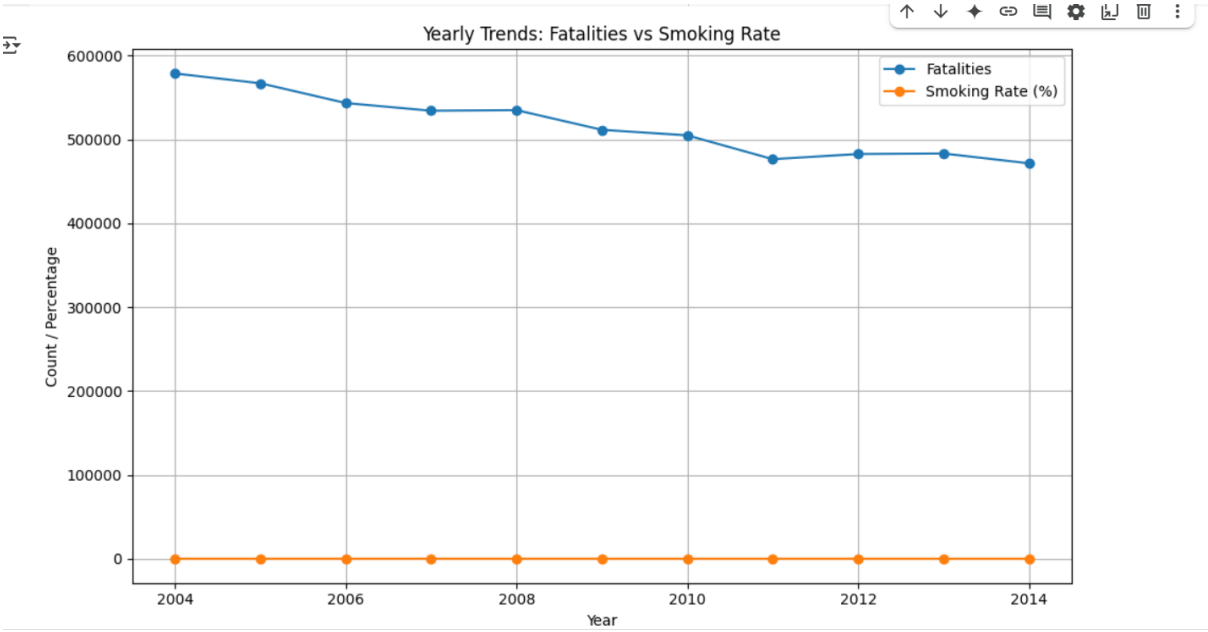
4.14 Plot: Time Series of Fatalities and Smoking Rate

Plot: Time Series of Fatalities and Smoking Rate

```
plt.figure(figsize=(10, 6))
plt.plot(df['Year'], df['Fatalities'], marker='o', label='Fatalities')
plt.plot(df['Year'], df['Avg_Smoking_Rate'], marker='o', label='Smoking Rate (%)')
plt.title("Yearly Trends: Fatalities vs Smoking Rate")
plt.xlabel("Year")
plt.ylabel("Count / Percentage")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

This Code Explains:

- Fatalities over time (in blue),
- Avg_Smoking_Rate over time (in orange),
- Both using markers 'o' for visibility,
- The x-axis is labeled "Year", and the y-axis is labeled "Count / Percentage".



Result:

- Fatalities (blue line) show a clear downward trend from 2004 to 2014.
- Smoking Rate (%) (orange line) also appears to remain flat or slightly decreasing (though hard to see due to scale).

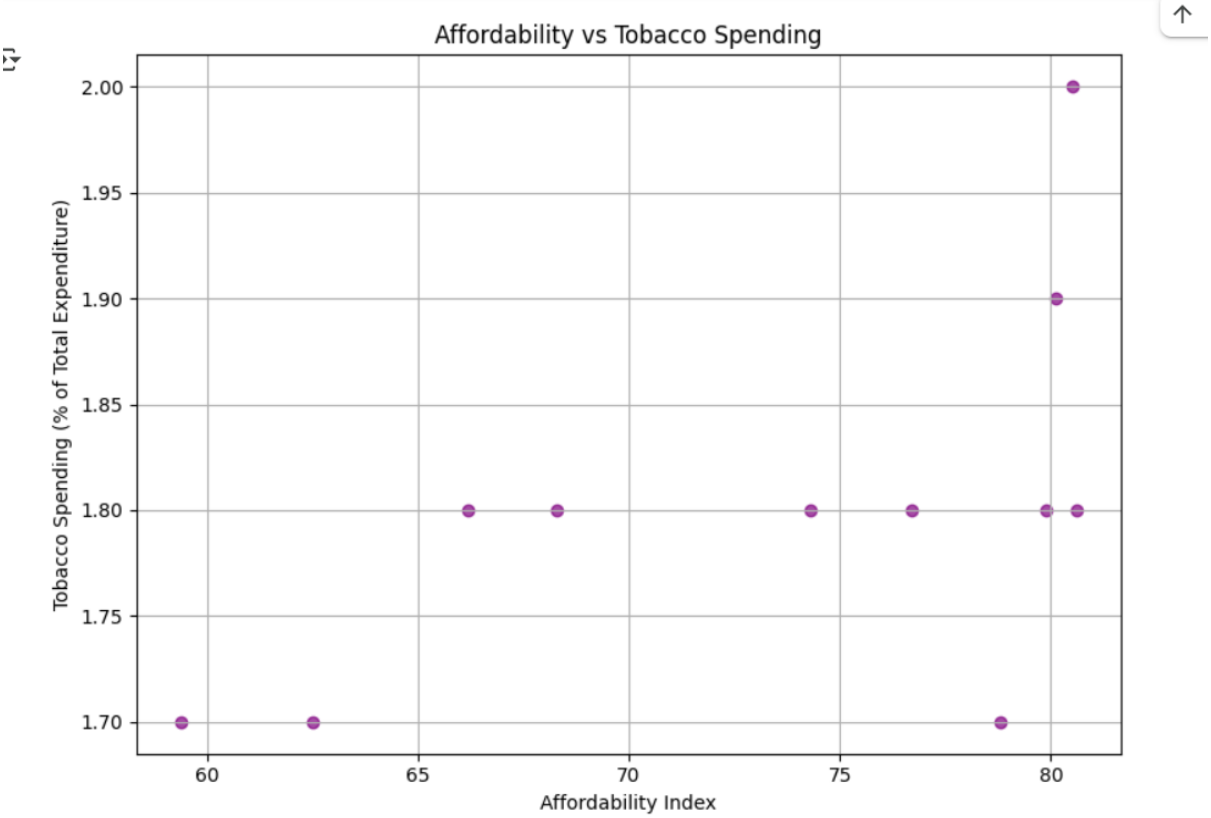
4.15 Plot: Tobacco Spending vs Affordability Index

Plot: Tobacco Spending vs Affordability Index

```
plt.figure(figsize=(8, 6))
plt.scatter(df['Affordability_Index'], df['Tobacco_Spending_Percent'], c='purple', alpha=0.7)
plt.xlabel("Affordability Index")
plt.ylabel("Tobacco Spending (% of Total Expenditure)")
plt.title("Affordability vs Tobacco Spending")
plt.grid(True)
plt.tight_layout()
plt.show()
```

This Code Explains:

- Sets the plot size to 8 inches by 6 inches.
- plots dots where the x-axis is the Affordability Index and the y-axis is Tobacco Spending (%), with:
    - c='purple': purple colored dots
    - alpha=0.7: 70% opacity (slightly transparent)
- adds descriptive labels and a title to make the plot understandable.
- Adds background grid lines to make reading values easier.



Result:

- **X-axis (Affordability Index)**: Higher values likely indicate that tobacco products are more affordable.
- **Y-axis (Tobacco Spending %)**: Shows how much of total spending goes to tobacco.

4.16 Correlation Heatmap

```
import seaborn as sns

plt.figure(figsize=(10, 8))
corr_matrix = df[features + [target]].corr()
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm", linewidths=0.5)
plt.title("Correlation Matrix")
plt.tight_layout()
plt.show()
```

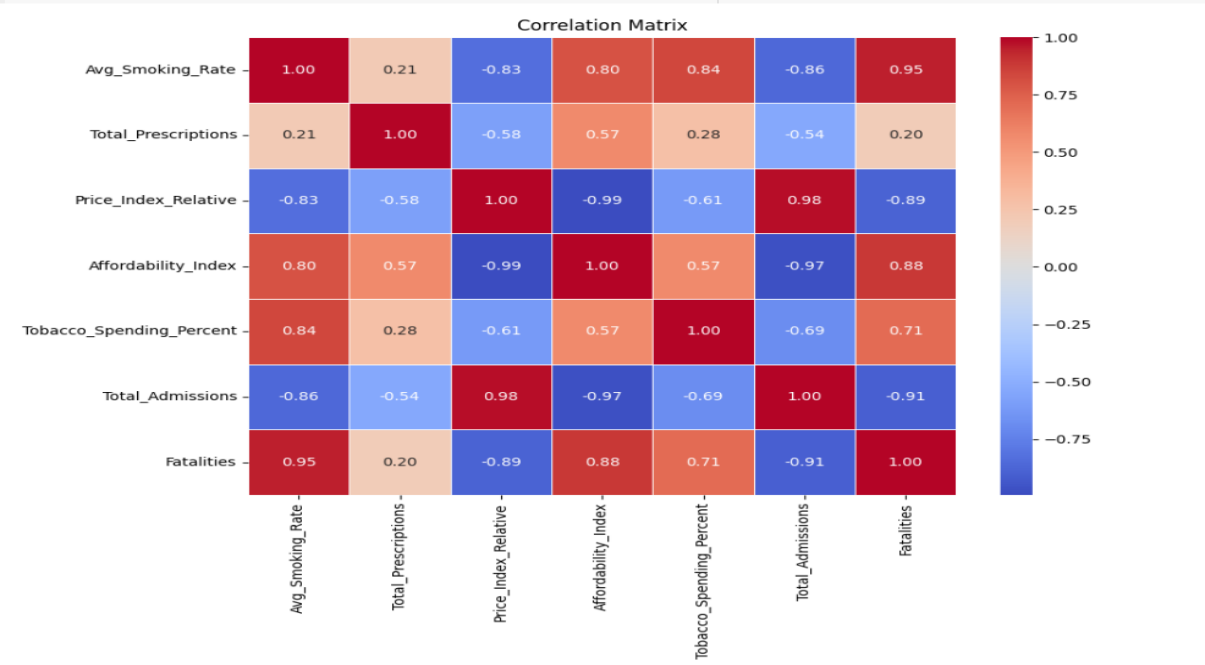
This Code Explains:

```
corr_matrix = df[features + [target]].corr()
```

- This calculates the Pearson correlation coefficients between all numerical columns in the combined list of features and target.

```
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm", linewidths=0.5)
```

- This creates a heatmap where:
  - **annot=True:** shows the correlation values inside the cells.
  - **fmt=".2f":** limits the numbers to 2 decimal places.
  - **cmap="coolwarm":** uses a red-blue gradient: red for positive correlations and blue for negative ones.
  - **linewidths=0.5**: separates the cells visually.



Result:

- Each cell shows the correlation between two variables:
  - +1 means perfect positive correlation (as one increases, so does the other).
  - -1 means perfect negative correlation (as one increases, the other decreases).
  - 0 means no linear correlation.
- Tobacco_Spending_Percent has:
  - Strong positive correlation with Avg_Smoking_Rate (0.84)
  - Moderate positive correlation with Affordability_Index (0.57)
  - Moderate negative correlation with Total_Admissions (-0.69)
- Affordability_Index is:
  - Almost perfectly negatively correlated with Price_Index_Relative (-0.99) — this makes sense: higher price → lower affordability.
  - Strongly positively correlated with Avg_Smoking_Rate (0.80) and Fatalities (0.88)
- Avg_Smoking_Rate is:
  - Very strongly negatively correlated with Total_Admissions (-0.86) — this may need further context for interpretation.
  - Very strongly positively correlated with Fatalities **(**0.95**)** and Tobacco_Spending_Percent **(**0.84).

5. Key Insights

- **Smoking Prevalence Drives Fatalities**
  - Avg_Smoking_Rate is the most influential factor in predicting tobacco-related deaths.
  - It shows a very strong correlation with fatalities (0.95)**.**

    - Feature importance analysis in the XGBoost model confirms this, as it's the dominant predictor**.**
- **Economic Factors Matter, But Less**
    - Affordability_Index and Price_Index_Relative are strongly correlated (almost perfect inverse: -0.99), confirming economic affordability is a significant aspect of smoking behavior.
    - Tobacco_Spending_Percent has moderate correlations with Affordability_Index (0.57) and Avg_Smoking_Rate (0.84), indicating affordability influences consumption but not as dominantly as smoking prevalence.
- **Healthcare Impact Is Linked**
    - Total_Admissions due to smoking-related illnesses negatively correlate with Avg_Smoking_Rate (-0.86) and Affordability_Index (-0.97), which may reflect healthcare interventions or reporting differences as smoking declines.
- **Model Performance**
    - The XGBoost model explains 83.5% of the variance in smoking-related fatalities ($R^2 = 0.835$), indicating strong predictive capability.
    - Residual analysis shows some large errors, suggesting model limitations with extreme values .
    - The model tends to overpredict or underpredict a few values significantly.
- **Trends Over Time**
    - Fatalities have steadily decreased from 2004 to 2014**.**
    - Smoking rate also shows a slow but downward trend, likely due to public health campaigns, regulations, and cessation support.

6. Conclusion

- Smoking behavior remains the single biggest predictor of tobacco-related deaths. Reducing smoking prevalence through targeted interventions should remain the highest policy priority.
- Affordability and economic access to tobacco products contribute to consumption, but their effect is secondary to smoking rates themselves.
- Government efforts like prescriptions and taxation can indirectly help, but their role appears smaller in the predictive model.
- Model-based prediction tools like XGBoost can provide strong estimates of future fatalities and support policy planning, especially when paired with rich, clean datasets.
- Policymakers should focus on reducing smoking prevalence directly, including education, regulation, and smoking cessation support, while using affordability and economic levers to reinforce behavior change.